NAME

ffortid – in dtroff output, find and reverse all text in designated right-to-left fonts, carry out stretching in Arabic, Hebrew, and Persian text, and arrange that words in slantable fonts are printed on a slanted base line.

SYNOPSIS

ffortid [–**r***font-position-list*] [–**w***paperwidth*] [–**-***font-position-list*] ...
[–**s**[**n**|[[**l**|**c**|**e**|**b**][**f**|**2**|**m**[*amount*]|**a**|**ad**|**al**]]] [–**ms**[**c**|**l**|**w**] ...

DESCRIPTION

ffortid's job is to take the *dtroff*(1) output which is formatted strictly left-to-right, to find occurrences of text in a right-to-left font and to rearrange each line so that the text in each font is written in its proper direction. *ffortid* deals exclusively with *dtroff* output, it does not know and does not need to know anything about any of *dtroff*'s preprocessors. Therefore, the results of using *ffortid* with any of *dtroff*'s preprocessors depends only on the *dtroff* output generated as a result of the input to *dtroff* from the preprocessors. Furthermore, the output of *ffortid* goes on to the same device drivers to which the *dtroff* output would go; therefore, *ffortid*'s output is in the same form as that of *dtroff*. *ffortid* reads its input from the standard input and write to the standard output.

In the command line, the –**r***font-position-list* argument is used to specify which font positions are to be considered right-to-left. A *font-position-list* is a list of font positions separated by white space, but with no white space at the beginning. *ffortid*, like *dtroff*, recognizes up to 256 possible font positions (0-255). The actual number of available font positions depends only on the typesetting device and its associated *dtroff* device driver. The default font direction for all possible font positions is left-to-right. Once the font direction is set, either by default or with the –**r***font-position-list* argument, it remains in effect throughout the entire document. Observe then that *ffortid*'s processing is independent of what glyphs actually get printed for the mounted fonts. It processes the designated fonts as right-to-left fonts even if, in fact, the alphabet is that of a left-to-right language. In fact, it is possible that the same font be mounted in two different positions, only one of which is designated as a right-to-left font position. This is how a single font can be printed left-to-right and right-to-left in the same document.

In addition to the specified font directions, the results of *ffortid*'s reformatting also depends on the document's *current formatting direction*, which can be either left-to-right or right-to-left. The default formatting direction is left-to-right and can be changed by the user at any point in the document through the use of the

```
x X PL
```

and

```
x X PR
```

commands in the *dtroff* output. These commands set the current formatting direction to left-to-right and right-to-left, respectively. These commands are either meaningless or erroneous to *dtroff* device drivers; therefore they are removed by *ffortid* as they are obeyed. These commands can be generated by use of

```
\X'PL'
```

and

```
\X'PR'
```

escapes in the *dtroff* input. For the convenience of the user, two macros

```
.PL
```

and

.PR

are defined in the **–mX2** and **–mXP** macro packages, that cause generation of the proper input to *ffortid*. They are defined by

```
..de PL
\\X'PL'
..
.de PR
\\X'PR'
..
```

If the current formatting direction is left-to-right, all formatting, filling, adjusting, indenting, etc. is to appear as occurring from left to right. In each *dtroff* output line, any sequence of contiguous right-to-left font characters is reversed in place.

If the current formatting direction is right-to-left, all formatting, filling, adjusting, indenting, etc. is to appear as occurring from right to left. Each *dtroff* output line is reversed, including both the left and right margins. Then, any sequence of contiguous left-to-right font characters is reversed in place.

The **–w***paperwidth* argument is used to specify the width of the paper, in inches, on which the document will be printed. As explained later, *ffortid* uses the specified paper width to determine the width of the right margin. The default paper width is 8.5 inches and like the font directions, it remains in effect throughout the entire document.

It is important to note that *ffortid* uses the specified paper width to determine the margin widths in the reformated output line. For instance, suppose that a document is formatted for printing on paper 8.5 inches wide with a left margin (page offset) of 1.5 inches and a line length of 6 inches. This results in a right margin of 1 inch. Suppose then that the text specifies a current formatting direction of right-to-left. Then, *ffortid*'s reformatting of the output line results in left and right margins of 1 and 1.5 inches, respectively. This margin calculation works well for documents formatted entirely in one direction. However, as a document's formatting direction changes, the resulting margins widths are exchanged. Thus, **.PL**'s right and left margins end up not being the same as **.PR**'s right and left margins. The user can make *ffortid* preserve the left and right margins by specifying, with the **–w***paperwidth* argument, a paper width other than the actual paper width. This artificial paper width should be chosen so that both margins will appear to *ffortid* to be as wide as the desired left margin. For example, for the document mentioned above, a specified paper width of 9.0 inches results in reformatted left and right margins of 1.5 inches each. The resulting excess in the right margin is just white space that effectively falls of the edge of the paper and does not effect the formatting of the document.

There is one exception to these simple rotation rules in that *ffortid*, at present, makes no attempt to reverse any of *dtroff*'s drawing functions, such as those used by *pic*(1) and *ideal*(1) (which are also available directly to the user). It is therefore suggested that these drawing functions, and thus *pic* and *ideal*, be used only when the current formatting direction is left-to-right. Additionally, due to the cleverness of the *dtroff* output generated by most substantial *eqn*(1) equations, it is suggested that *eqn*'s use also be limited to a left-to-right formatting direction for all but the simplest forms of equations. These rules do not in any way restrict the use of right-to-left fonts in the text dealt with by any of the preprocessors, but simply suggest that these particular preprocessors be used only when the formatting direction is left-to-right.

An additional point to keep in mind when preparing input both for *dtroff*'s preprocessors and for *dtroff* itself is that *ffortid* rotates, as a unit, each sequence of characters of the same direction. In order to force *ffortid* to rotate parts of a sequence independently, as for a *tbl*(1) table, one must artificially separate them with a change to a font of the opposite direction.

The **–-***font-position-list* argument is used to indicate which fonts positions, generally a subset of those designated as right-to-left, contain fonts for Arabic, Hebrew, Persian, or related languages, whose fonts support stretching of letters and/or connections. For these fonts, left and right justification of a line can be achieved by stretching instead of inserting extra white space between the words in the line. If requested by use of the **–s** argument described below, stretching is done on a line only if the line contains at least one word in a **–-** designated font. If so, stretching is used in place of the normal distributed extra white space insertion for the entire line. The intention is that stretching soak up all the excess white space inserted by *dtroff* to adjust the line. If there are no opportunities for stretching or there are too few to soak up all the excess white space, what is not soaked up is distributed in between the words according to *dtroff*'s algorithm. There are several kinds of stretching, and which is in effect for all **–-** designated fonts is specified with the **–s** argument, described below. If it is desired not to stretch a particular Arabic, Hebrew, Persian, or other font, while still stretching others, then the particular font should not be listed in the **–-***font-position-list*. Words in such fonts will not be stretched and will be spread with extra white space if the containing line is spread with extra white space.

The **–r** and the **–-** specifications are independent. If a font is in the **–-***font-position-list* but not in the **–r***font-position-list*, then its text will be stretched but not reversed. This independence can be used to advantage when it is necessary to designate a particular Arabic, Hebrew, Persian, or other font as left-to-right for examples or to get around the above mentioned limitations in the use of *eqn*, *ideal*, *pic*, or *tbl*.

The kind of stetching to be done for all fonts designated in the **–-***font-position-list* is indicated by the **–s** argument. There are two relatively independent dimensions that must be set to describe the stretching, what is stretched and the places that are stretched. A stretch argument is of the form

–s*mp*
or
–sn

where *m* specifies the stretching mode, i.e, what is stretched, and *p* specifies the places that are stretched. The *m* and *p* must be given in that order and with no intervening spaces. The **–sn** means that there is *no* stretching and normal spreading of words is used even in **–-** designated fonts. The choices for the mode *m* are:

l (letter ell)
In the words designated by the *p*, stretch the last stretchable letter.

c
In the words designated by the *p*, stretch the last connection to a letter.

e
In the words designated by the *p*, stretch either the last stretchable letter or the last connection to a letter, whichever comes later.

b
In the words designated by the *p*, stretch either the last stretchable letter or the last connection to a letter, whichever comes later, and if it is a letter that is stretched and it is a connect-previous letter then also stretch the connection to the letter.

Not all modes are available for all fonts. For example, fonts for Hebrew, whose letters are not connected do not support connection stretching. While Arabic, Hebrew, and Persian traditionally do have letter stretching, not all fonts for them support letter stretching. *ffortid* attempts to stretch all **–-** designated fonts in the specified modes, but in any text, ends up doing only those stretches that are possible given in the text's current font. To allow *ffortid* to know what stretches are possible, the width tables for stretchable fonts have some additional lines that must come somewhere after the **name** line and before the **charset** line.

```
stretchable: letters connections
stretchable: connections letters
```

```
stretchable: connections
stretchable: letters
```

Each such stretchable font must have one of the first four lines. We now discuss the various ways that different kinds of stretch are achieved in the available fonts and how *ffortid* deals with each.

To our knowledge, all Arabic and Persian fonts, have a baseline filler that can be used to achieve the stretching of connections. It is fairly easy for such a filler to be added to any font definition that does not have it, and moreover to make it the character that is addressed by **\(hy**, which is normally the code for the hyphen character. Since Arabic and Persian do not have a hyphen and hyphenation is turned off when in an Arabic or Persian font, it is safe to use **\(hy** to name the filler. Of course, this requires that the width table for Arabic and Persian fonts have an entry for **hy** designating the filler character in the font, for example:

```
hy    15 0 0267   filler
```

Giving the filler character an explicit *dtroff* two-character name allows *dtroff* to deal with it uniformly despite that it might be in a different position in each font.

On the other hand, stretching of letters requires a dynamic font which, by its very nature of not having a constant bitmap for a given font name, point size, and character name, cannot be type 1 (in PostScript terminology) and cannot be a bitmapped font. Therefore, as mentioned, not all Arabic, Hebrew, and Persian fonts support stretching of letters. Moreover, within a dynamic font, not all characters are stretchable. Historically, only characters with strong horizontal components are stretchable, such as those in the stand-alone and connect-previous forms of the *baa* family. Obviously, one cannot stretch totally vertical characters such as *alif*. Therefore, it is necessary to specify by additional information in the *dtroff* width table for a font which characters are stretchable. In the width table for an Arabic, Hebrew, or Persian font, for each character, one specifies after the name, width, ascender-descender information, and code, two additional fields, the connectivity and the stretchability of the character, in that order. The connectivity is either

N for stand-alone,
A for connect-after,
P for connect-previous,
B for connect-both, or
U for unconnected (because it is punctuation or a diacritical, etc.),

and the stretchability is either

N for not stretchable,
S for stretchable,

Some examples of width table lines are:

```
%     125 2 045   percent

---   55 0 0101   U     N     comma
---   70 0 0105   U     N     hamza

---   129 0 0106  N     S     baa_SA
---   36 2 0102   N     N     alef_SA

---   113 0 0177  A     N     sad_CA
---   66 2 0215   A     S     caf_CA

---   43 2 0225   P     N     alef_CP
```

```
---   120 0 0274  P     S     baa_CP

---   53 0 0230   B     N     baa_CB
---   73 2 0261   B     S     caf_CB
```

Recall that `---` in the name field of a character means that it can be addressed only by **`\N'`***`n`*`'`, where *n* is the decimal equivalent of the character's code.

For a Hebrew font, for which there is no notion of connectivity of letters, and therefore, the position of the letters is irrelevant for deciding stretching, there is only the possibility of stretching letters. Some examples of width table lines for such fonts are:

```
%     132 3 045   percent

---   95 3 0140   U     N     quoteleft=alef
---   92 3 0141   U     S     a=bet
```

In a dynamic font, there are two additional, alternative ways that stretching of connections can be achieved.

- The filler is a stretchable letter, normally of width zero, to which the total width of the filler is passed as the stretch amount.
- The connecting portions of all connecting letters are themselves stretchable in the same way as the stretchable letters are. In this situation to achieve a total connection stretch of *x*, one would pass *x*/2 to each of the connecting-after portion of the before letter and the connecting-before portion of the after letter.

The use of the first of these solves the problems caused by the fact that amount of a given connection stretch may not be integrally divisible by the width of the filler. A stretchable filler can be stretched to any amount. The use of the second improves the appearance of the connection stretch. While letter stretching is done with nice, smooth curves, connection stretching using the very straight filler is noticeably flatter and there are observable corners where the filler meets the generally curved connecting parts of its adjacent letters. While the fixed-size filler seems to be available on all Arabic and Persian fonts, stretchable fillers and stretchable connecting parts are available only with type 3 PostScript fonts, although it would be possible to provide a stretchable filler as the only locally defined character in a type 3 font that falls to another type 1 font for all the other characters, which are only virtual in the type 3 font.

The *dtroff* width table for any font providing a stretchable filler or stretchable connecting parts must have an additional line to specify the nature of the connection stretch in the font, which must be one of the following.

```
connection stretch: fixed filler
connection stretch: stretchable filler
connection stretch: stretchable connections
```

This line must come somewhere after the **`name`** line and before the **`charset`** line. If none is specified, it is assumed to be the first. Therefore, it is not necessary to say anything new for the typical type 1 or bitmapped font with a fixed-sized filler. Note that if a font allows different kinds of connection stretching, only one can be specified per mounting of the font specified in a single width table. If one wants to use the same font with different ways of stretching connections, one must mount the same font under different names in different width tables, each specifying a different kind of connection strecthing.

ffortid implements the connection stretching that is requested by the **`-s`** command-line arguement as well as it can using the kind of connection stretching available for the font being used. Thus, if one is not using fixed-sized fillers, *ffortid* ignores the various options put in to deal with the fact that an integral number of

fillers may not fulfill the needed stretch.

Below, "stretchable unit" refers to that which is a candidate for stretching according to the mode. The choices for *p*, which specifies places of stretching, are:

f

In any line, stretch the last stretchable unit.

2

Assuming that the mode is **b** (both), in any line, stretch the last two stretchable units, if they are the connection leading to a stretchable connect-previous letter and that letter, and stretch only the last stretchable unit otherwise. If the mode is not **b**, then this choice of places is illegal.

m*n* or **m**

In any line, stretch the last stretchable unit by an amount not exceeding *n* emms. If that does not exhaust the available white space, then stretch the next last stretchable unit by an amount not exceeding *n* emms, and so on until all the available white space is exhausted. If *n* is not given, it is assumed to be **2.0**. In general *n* can be any number in floating point format.

a, **ad**, or **al**

In any line, stretch all stretchable units. In this case, the total amount available for stretching is divided evenly over all stretchable units on the line identified according to the mode. Since the units of stretching are the units of device resolution, the amount available might not divide evenly over the number of places. Therefore, it is useful to be able to specify what to do with the remainder of this division. This specification is given as an extension of the stretching argument. The choices are **d** or **l**, with the former indicating that the excess be distributed as evenly as possible to the spaces between words and the latter indicating that the excess be distributed as evenly as possible in stretchable letters that were stretchable units according to the current mode and place. The latter is the default if no choice is specified. The stretched item for the **l** choice must be a letter rather than a connection because only a stretchable letter is stretchable to any small amount that will be the remainder. Of course, if the method of stretching a connection is dynamic, then a connection could be stretched to any amount, but then there would not be a remainder in the first place.

In general, the stretch is divided as evenly as possible between all stretchable units in a line. Specificly, in stretch mode **b**, if we have a connection leading to a stretchable connect-previous letter and that letter, then any stretch remainder we have from stretching the connection will be added to the stretch of the letter.

Sometimes, it is desirable to be able to manually stretch connections or letters to achieve special effects, e.g., more balanced stretching or stretching in lines that are not otherwise adjusted, e.g., centered lines.

If fixed-sized fillers are used to achieve connection stretching, then one can use the filler character explicitly as many times as necessary to achieve the desired length. Note that the *troff* line drawing function can be used to get a series of adjacent fillers to any desired length, e.g.,

```
\l'2m\(hy'
```

will draw a string of adjacent baseline fillers of length 2 emms.

How to manually stretch connections that are done by a stretchable filler or by stretchable connection parts is described after describing how to manually stretch letters themselves.

To achieve stretching of letters, one should immediately preceed, with no intervening printable text, the letter to be stretched by the escape sequence

```
\X'stretch'\h'n'
```

where *n* is a valid length expression in *troff*'s input language. *ffortid* is prepared to deal with the output from *dtroff* generated by this input to generate output that will cause the letter immediately following it to be stretched by the length specified in *n*. For example,

\X'stretch'\h'1m'\N'70'

will cause the character whose decimal code is 70 to be stretched by 1 mm. The output will fail to have the desired effect if the letter following is not a stretchable letter.

If connection stretching is achieved by having a stretching filler, then one manually stretches the filler character by the desired amount as if it were a letter.

\X'stretch'\h'*n***'\(hy**

Here, though the stretch parameter *n* is the total length of the filler, as the filler is of length zero if it is not stretched.

To stretch the connecting parts of letters, two additional escape sequences are provided that may be placed before, with no intervening printable text, the letter to which they apply,

\X'BCstretch'\h'*nb***' \X'ACstretch'\h'***na***'**

where *nb* and *na* are valid length expressions in *troff*'s input language. These specify the amounts of stretch in the before and after connecting parts of the immediately following letter. The order in which the **\X'stretch'\h'***n***'**, **\X'BCstretch'\h'***nb***'**, and **\X'ACstretch'\h'***na***'** for a letter appear is irrelevant, but in between them and after the last of them, there is *no* printable text, including white space (including new lines), and the letter to which they apply immediately follows the last. Suppose that two consecutive, in logical order, letters have decimal codes 70 and 80. Suppose also that 70 connects after to the connecting before 80. Suppose finally that this connection from 70 to 80 is to be stretched by 1 emm and the letter 80 is to be stretched by 2 emms. Then the input would look as follows:

\X'ACstretch'\h'.5m'\N'70'\X'BCstretch'\h'.5m'\X'stretch'\h'2m'\N'80'

Note that the connection stretch of 1 emm was split into two stretches of .5 emm for each of the connecting after and the connecting before parts.

For finer control over stretching, it may be desirable to inhibit automatic stretching on manually stretched connections and letters. In particular, when manual stretching is done on a letter or its connection for balancing purposes, one does not want additional automatic stretching to be done on the same to mess up the balance. Accordingly, three command line flags are provided for this purpose:

–msc
: Do not automatically stretch manually stretched connections.

–msl
: Do not automatically stretch manually stretched letters.

–msw
: Do not automatically stretch any word containing any manual stretching.

These flags are understood as eliminating potential stretching places, letters or connections, that were identified on the basis of the stretch mode, **l**, **c**, **e**, or **b**. (In the following description, parenthesized text is a comment stating what is true at this point and not what needs to be done.)

For any letter *l* that is a candidate for stretching by the mode,

> **if** *l* is in a word containing a manually stretched letter or connection and **–msw** is set, **then** *l* is no longer a candidate
>
> **otherwise**
>
> **if** both the letter itself and its connection to the previous letter are candidates **then**

if either kind of manual stretch is in the letter and that kind of manual stretch cannot be stretched additionally, **then** neither part of *l* is any longer a candidate;

otherwise (only the letter itself is a candidate OR only its connection to the previous letter is a candidate)

if the letter itself is a candidate for stretching by the mode,

if there is manual stretching in the letter and manually stretched letters cannot be stretched more, **then** *l* is no longer a candidate;

otherwise (the connection of *l* is a candidate for stretching by the mode),

if there is manual stretching in the connection of *l* to the previous letter and manually stretched connections cannot be stretched more, **then** *l* is no longer a candidate.

ffortid is able to arrange that text in slantable fonts is printed with each word in a line of text in a slanted baseline that crosses the baseline of the line at the center of the word. The figure below shows each words baseline as a solid arrow and the line's baseline as a dotted arrow.

figure baselines.ps

Observe that in this style of printing the beginning of a non-first word is directly over the end of its previous word. Moreover, within a word there will generally be stretching to allow this property to hold; that is, if there were no stretching to achieve left justification, it might be necessary to have a horizontal gap between two consecutive words.

For *ffortid* to implement this slanted-baseline printing for a font, it is necessary that some non-standard information be supplied in the *dtroff* width table for the font. First, there is a line that specifies the slant in degrees.

```
slant 22.0
```

The argument can be a floating point number. This line must come somewhere after the **`name`** line and before the **`charset`** line. The argument should be the slant in degrees and should match the slant implied by the first two values in the **`FontMatrix`** of the font. Specifically the ratio of the second to the first should be the tangent of the slant. *ffortid* uses this slant value to know by how much to displace the beginning of a word vertically so that as it flows downward in the right-to-left direction, the center of the word crosses the line's baseline.

In addition, in order that there appear to be no horizontal white space between words, the **`spacewidth`** of the font must be set to one.

```
spacewidth 1
```

Actually, the spacewidth should be zero, but *dtroff* refuses to set the it to zero, setting it to an emm width if you tell it zero. To the human eye, at the typical resolutions specified in the *DESC* files, in the mid hundreds, a spacewidth of one is close enough.

A few suggestions to the user are in order. While *dtroff* supports font changes in the middle of words, *ffortid* does not support and reports as an error font changes that change the slant in the middle of words, either to another nozero slant or to no slant at all. Besides it being a pain to implement, it is not clear what the behavior should be in such a situation. Recall also that there is typically no horizontal separation between slanted words; all the separation comes from the end of one word being separated vertically from the beginning of the next. If words are too short, there may not be enough vertical clearance between consecutive words. To insure adequate vertical clearance, it may be necessary to combine several words into what *dtroff* and *ffortid* consider one word. For this purpose, each such slantable font should have a special

character called **\(ps** (for “permanent space”, whose width is set to what would normally be the spacewidth and which can be used as an unpaddable blank between two words that are to be treated as a single, unbreakable word by *dtroff* and *ffortid*.

```
ps    72 0 040    permanent space
```

Note that the normal *dtroff* unpaddable space, “**\ **”, cannot be used, because its width is defined to be that of the regular space, i.e., the spacewidth, and would end up being one in this case. If one wants the guaranteed white space, but wants to allow a word break, one can make the **\(ps** the last character or the first character in a regular, white-space-delimited word.

FILES

/usr/lib/tmac/tmac.*	standard macro files
/usr/lib/font/dev*/*	device description and font width tables

SEE ALSO

Cary Buchman, Daniel M. Berry, *User’s Manual for Ditroff/Ffortid, An Adaptation of the UNIX Ditroff for Formatting Bi-Directional Text,*
Johny Srouji, Daniel M. Berry, *An Adaptation of the UNIX Ditroff for Arabic Formatting*
troffort(l), ptrn(l)